# QUANTUM OPTIMIZATION FROM A COMPUTER SCIENCE PERSPECTIVE

An Undergraduate Research Scholars Thesis

by

DARRYL CHERIAN JACOB

Submitted to the Undergraduate Research Scholars Program at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by Research Advisor: Dr. Fang Song

May 2020

Major: Computer Science

# TABLE OF CONTENTS

# ABSTRACT

Quantum Optimization From A Computer Science Perspective

Darryl Cherian Jacob
Department of Computer Science and Engineering
Texas A&M University

Research Advisor: Dr. Fang Song
Department of Computer Science and Engineering
Texas A&M University

Optimization problems are ubiquitous in but not limited to the sciences, engineering, and applied mathematics. Examples range from the fastest way USPS can route packages through a delivery network to the best way an autonomous vehicle can navigate through a given traffic environment. Classical optimization algorithms dominate the way we solve these problems. However, with the rapid advance of quantum computers, we are looking at novel, quantum-inspired ways of solving old problems to achieve some speedup over classical algorithms. Specifically, we are looking at the Quantum Approximate Optimization Algorithm (QAOA). We show that QAOA provides a tunable, optimization algorithm whose quantum circuit grows linearly with the number of constraints for MAXSAT, an NP-complete problem.

# DEDICATION

I dedicate this thesis to my family and friends. There is no point in gaining knowledge if not to improve your lives.

# ACKNOWLEDGMENTS

The person who deserves the most credit for the completion of this thesis is my faculty advisor, Dr. Fang Song. Without your guidance, my aimless rambling would be my downfall.

I would also like to thank HGK and the Computational and Data Intensive Physics Group at A&M. You provided the foundation for my research experience and I know my work is enriched because of my time with you.

Finally, thanks to my father for his amazing example and relentless support. I am equally grateful to my mother for tolerating my dumbest ideas and encouraging my best ones.

# CHAPTER I

# INTRODUCTION

## Optimization

*What is a Problem?*

In Computer Science, problems are viewed as mappings from an input to an output set. Formally, [1] states the well defined problem "specifies in general terms the desired input/output relationship". Such a vague definition can encompass, arguably, all problems one might face. One surprisingly relevant problem is finding the fastest way back home from work. In "general terms", the inputs are starting and final locations (work and home, respectively). The output is the fastest path from starting to final location. For every pair of starting and final locations (ignoring traffic, tolls, gas, etc.), there exists a fastest route and the problem wants us to find it.

So what is 'fast'? It is an essential component in our "desired relationship". Perhaps, 'fast' is 'within $T$ minutes', where $T$ is some number. Instead, 'fast' might be 'as little time as possible'. The former definition specifies a bound or decision within which we will be satisfied i.e. there is no incentive to use less than $T$ minutes. The latter uses time as a performance measure (or objective function); when this measure is minimized, our satisfaction is maximized. The former is a 'decision' problem. The latter is an 'optimization' problem.

A 'decision' problem seeks a relationship between the inputs and whether a particular criterion, or 'decision', is satisfied i.e. did we take less than $T$ minutes? An 'optimization' problem seeks to minimize this criterion i.e. lowest $T$ we can possibly achieve. A decision problem can be converted into an optimization problem. We do this by parameterizing the decision using $T$, where $T$ is the desired performance measure ex. 5 minutes. Similarly, Optimization problems can be converted to decision problems. Any problem can be formulated as a decision or optimization problem. The relationship between the central concept of a problem and the classifications is illustrated in Figure 1.

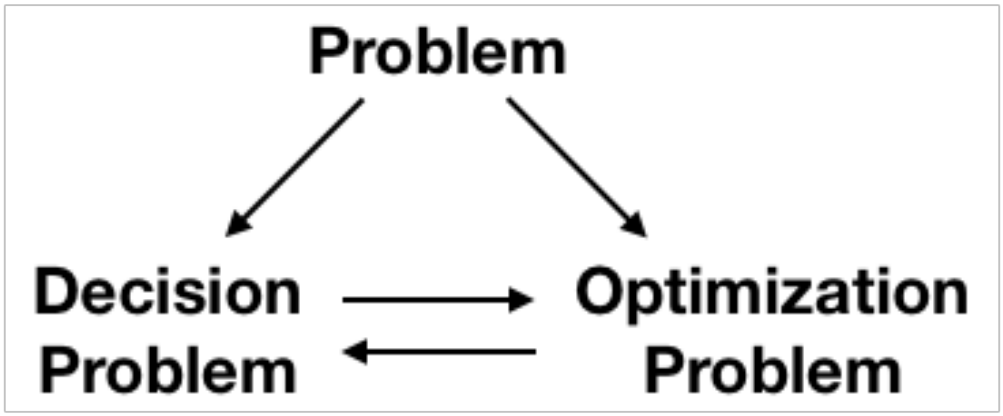


Figure 1: Relationship between Problem Classifications

*MAXSAT*

MAXSAT is one example of an optimization problem. It's corresponding decision problem is known as "satisfiability". MAXSAT presents a list of "clauses", or conditions, over some boolean variables and asks what assignment of zeros and ones satisfies the most number of clauses. If there are $m$ clauses, solutions would have an objective function value from 0, where no clauses are satisfied, to $m$, all clauses satisfied. "Satisfiability" asks if there is an assignment which satisfies all clauses.

Each clause can be seen as a disjunction, or boolean "OR", of many variables. If there are 3 variables, $\{x_0, x_1, x_2\}$, an example of a clause may be $x_0 \cup \neg x_1$, where $\neg x$ is the boolean "NOT" of the variable $x$. A satisfying assignment would be $x_0 = 1, x_1 = 0, x_2 = 0$ (or $100$) since $1 \cup \neg(0) = 1 \cup 1 = 1$. If we add another clause, say, $x_0 \cup x_1 \cup x_2$, $100$ would still be a satisfying assignment since $1 \cup 1 \cup 1 = 1$. Hence, for this particular instance with 3 variables and 2 clauses, $100$ would have an objective function value of 2.

**Classical Computing**

Digital logic is largely based on the logical manipulation of a bit, or binary digit. Everything is known about a given bit's state; it is either on (1) or off (0). By following the rules set in [2], it is possible for one to perform 'regular' operations like addition, division and multiplication on long

strings of bits. Gates represent the symbolic, fundamental operations that make up the 'regular' ones. This makes classical computers amazing predictable machines which, more often that not, produce the right answer. From automated industries to virtual assistants, the dizzying number of computer-related applications has led to soaring demand for more computing power. Figure 2 shows how demand impacted the supply of computing capability, a trend referred to as "Moore's Law". As computers grow faster, we can pose larger problems.

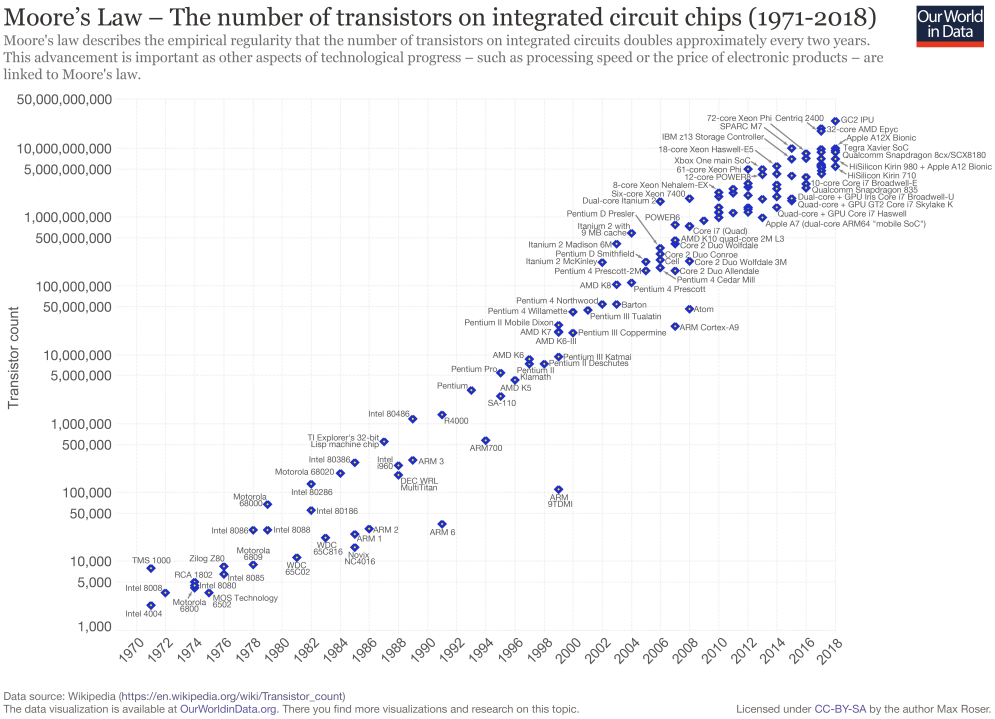


Figure 2: Growth in classical computing capability since 1971. Data from Roser, Max "Moore's Law Transistor Count 1971-2018", `https://commons.wikimedia.org/wiki/File:Moore%27s_Law_Transistor_Count_1971-2018.png`

### *P vs. NP*

As problems grow larger, the number of operations required to solve them grows larger. Withing this context, the relationship between a problem's size and the amount of time a method, or algorithm, uses to solve it becomes very important. Figure 3 demonstrate how drastic this

relationship can be. The polynomial and exponential relationships are especially relevant. Many problems in Computer Science can be solved in polynomial time. Some examples include linear programming and finding the shortest path between multiple locations. These are said to belong to the class, $P$. Other problems have not been demonstrated to be solvable in polynomial time; some may have solution times within the exponential regime. With a few more criteria, we say these problems belong to $NP$. When problems require an exponential amount of time to solve regardless of how much computing power, even small instances quickly become unfeasible. For example, finding groups in highly connected social networks (clique finding), optimal routes through multi-city tours (traveling salesman problem), and the smallest number of colors with which to color a map (graph coloring). This makes polynomial time algorithms for solving $NP$ problems a highly coveted, possibly impossible feat. For a more refined discussion on this problem hierarchy, please refer to [1].

Additionally, many $NP$ problems belong to the class $NP$-complete. All problems within $NP$-complete can be reduced, or converted, to one another. These include clique finding, the traveling salesman problem and graph coloring. Another example of an $NP$-complete problem is MAXSAT. Therefore, an algorithm to generate optimal solutions to MAXSAT can perform just as well for other $NP$-complete problems, an important step in the larger goal of solving all problems quickly. For the rest of this paper, we will consider QAOA applied to MAXSAT.

## Quantum Computing

*What can Quantum do?*

Given the interest in solving $NP$ problems, researchers are exploring new computing domains where faster algorithms might exist. One such domain, proposed by Dr. Richard Feynman, is Quantum Computing [3]. Here, in contrast to the classical case, the inner workings of a computer are not as predictable. The bit is replaced with the quantum bit, or qubit. Any interaction with the environment, even an observation, can interfere with the qubit's on/off state [4]. Similar to classical computers, gates are used to manipulate the qubit. Dealing with a unit of information that cannot be inspected until the end of a computation can be frustrating while looking for errors,

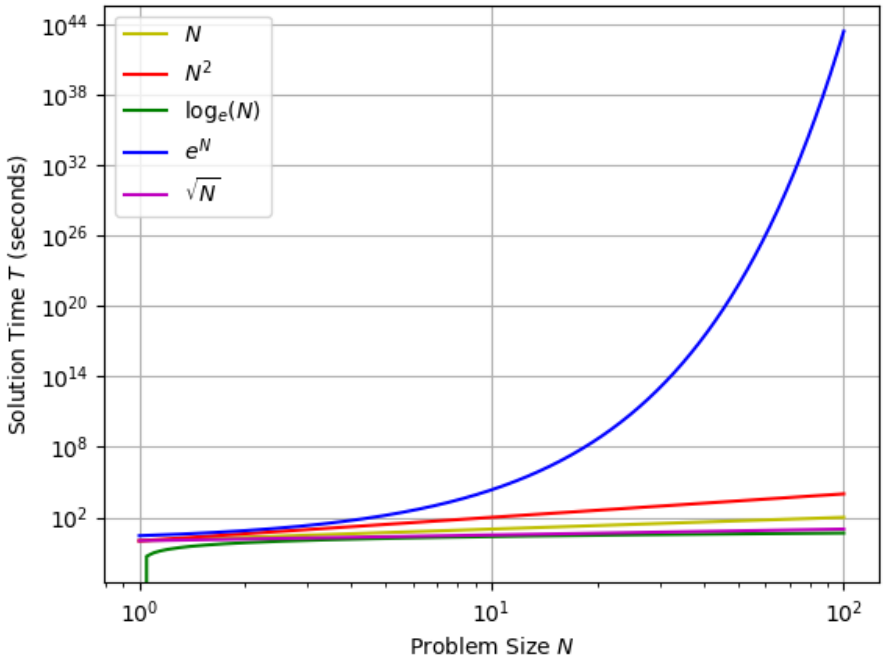


Figure 3: Growth in amount of time required to solve a problem $T$ as a function of problem size $N$, for various possible relationships. Y-Axis is logarithmic. Lower is better

but it makes certain phenomena like "tunneling" possible. These phenomena could help us solve $NP$ problems faster [5].

*Quantum Optimization*

For a large number of optimization problems, the algorithms require exponential time. For example, a string of $N$ binary digits can take $2^N$ different values. If each of these values was mapped to a number (say, by adding all the digits), one optimization algorithm requires we look through all $2^N$ values to find the the bit string which minimizes the given function. This is a seemingly contrived example but it can be extended. Interestingly, there is a quantum algorithm which can perform this search in $\sqrt{2^N} = 2^{\frac{N}{2}}$ operations [6]. This is a quadratic speedup and it is an open question whether more can be gained.

*Quantum Annealing*

Quantum Annealing is one example of a quantum optimization algorithm. Annealing is

where a metal is heated to high temperatures and cooled slowly, usually to increase hardness. Simulated Annealing is a classical algorithm which simulates heating and cooling over a graph, or lattice, to produce the lowest energy (or ground) state. The lowest energy state within a given graph could represent any quantity including, for a given MAXSAT instance, the boolean assignment which bears the lowest/highest objective function value. From [7], we know that Simulated Annealing will produce a global maxima if the system is cooled for an infinitely long time.

Quantum Annealing replaces heating and cooling with increasing and decreasing quantum fields which allows one to "tunnel" from one end of the solution space to a minima in another end. Since Quantum Annealing can be seen as a quantum analogue of Simulated Annealing, we are able to say that if the quantum, transverse fields are decreased infinitely slowly, the quantum system will evolve into a globally optimal solution. This evolution is from the ground state of a simple problem to that of the desired, complicated problem. The Quantum Approximate Optimization Algorithm can be viewed as a discretized version of Quantum Annealing; specific snapshots in the decreasing quantum field are used to reproduce the evolution from a simple to a complex ground state.

Within this thesis, we will explore new quantum optimization algorithms such as the Quantum Approximate Optimization Algorithm [8]. We could apply what we learn towards other domains which rely heavily upon optimization problems. I would like to undertake a theoretical proof for the workings behind QAOA and make statements on the optimality and growth in circuit depth (or running time) for an implementation of the algorithm. Specifically, if the tuning parameter for QAOA is increased, the quality of the sampled solution increases on average. Furthermore, similar to Quantum Annealing, if QAOA is run for infinitely long, a globally optimal solution will be returned. Additionally, the circuit depth for an implementation will grow linearly with the number of constraints for the given MAXSAT problem.

# CHAPTER II

# QUANTUM APPROXIMATE OPTIMIZATION ALGORITHM

In this chapter, I will explore the Quantum Approximate Optimization Algorithm proposed in [8]. I will provide formal definitions for concepts outlined in the paper as well as possible proofs for how QAOA may work.

**Algorithm**

**Algorithm 1:** Quantum Approximate Optimization Algorithm

**Data:** Number of Variables, $n$
Number of Constraints, $m$
Objective Function, $C : \{0,1\}^n \to \mathbb{Z}$ where $C(z) = \sum_{\alpha=1}^{m} C_\alpha(z)$
Quality of Approximation, $p \in \mathbb{Z}^+$
Stopping Criteria, $should\_stop$
**Result:** $z \in \{0,1\}^n$ such that $C(z)$ is close to $\max_{z^* \in \{0,1\}^n} C(z^*)$
**begin**
  $|z\rangle \leftarrow |0\rangle^n \in H$ ; `// Initialize return value`
  $c_z \leftarrow C(z)$;
  Choose $p$ angles in $[0, 2\pi]$, $\vec{\gamma} = \gamma_1, \ldots, \gamma_p$;
  Let $U(C, \gamma_i) = e^{-i\gamma_i C}$ ; `// Angle-dependent Unitary Operators`
  Choose $p$ angles in $[0, \pi]$, $\vec{\beta} = \beta_1, \ldots, \beta_p$;
  Let $B = \sum_{j=1}^{n} \sigma_j^x$;
  Let $U(B, \beta_j) = e^{-i\beta_j B}$;
  **while** $\neg should\_stop$ **do**
    $|s\rangle \leftarrow \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n} |i\rangle$ ; `// Superposition over all variables`
    $|\vec{\gamma}, \vec{\beta}\rangle \leftarrow U(B, \beta_p)U(C, \gamma_p) \ldots U(B, \beta_1)\, U(C, \gamma_1)|s\rangle$;
    Measure $|\vec{\gamma}, \vec{\beta}\rangle$ in the computational basis to get $|z_p\rangle$;
    $c_p \leftarrow C(z_p)$ ; `// Store best solution, z`
    **if** $c_p > c_z$ **then**
      $|z\rangle \leftarrow |z_p\rangle$;
      $c_z \leftarrow c_p$;
  **end**
  **return** $z$
**end**

This is how QAOA will run on a Quantum Computer. Our $n$-variable objective function $C$ is defined as a sum of smaller, constraining functions $C_\alpha$; if some bitstring (or state) $z$ satisfies $C_\alpha$,

$C_\alpha(z)$ is 1; otherwise, 0. Therefore, the maximum objective value $z$ can have with respect to $C$ is the number of constraints $m$. The procedure is tuned using a hyper-parameter $p$ which denotes the approximation quality i.e. bigger $p$, better solution. We start by initializing the output state to all zeros. The stopping criteria is some simple condition (ex. number of evaluations of objective function, running time, etc.) which represents when the algorithm should terminate. We wish to approximate the state $z$ which evaluates to the global maximum $\max_{z^* \in \{0,1\}^n} C(z^*)$ for $C$.

We start by setting the output state $z$ to all zeros $|0\rangle^n$. This is analogous to clearing memory before use. Our first solution and it's objective value $c_z$ are stored for later comparison. We pick $2p$ angles by some method; these angles directly affect the solution quality so the picking method must generate "good" angles $\vec{\gamma}, \vec{\beta}$ for a given problem. The angle-dependent operator $U(C, \gamma_i)$ denotes the imaginary exponential of the objective function. This is used to simulate (or time-evolve) the output state under the influence of the quantum-mechanical objective function. This is similar to the solution of the time-dependent Schrödinger equation:

$$\frac{d}{dt}|z\rangle = -iH|z\rangle$$
$$\therefore |z\rangle = e^{-iH|z\rangle}$$

where the Hamiltonian $H$ being simulated is the objective function $C$. $U(B, \beta_i)$ can be explained similarly as the simulation of an operator $B$; in this case, the operator is the sum of bit flip operators $\sigma_i$ i.e. $\sigma_i$ flips the $i$-th bit (or it's corresponding probability) of a state. The bitflip operator is denoted the Pauli-X operator.

If our stopping condition is unsatisfied, we begin our sampling procedure. We start with a uniform superposition over all states $s$. In a uniform superposition, if $s$ is measured, all $n$-length bitstrings from all zeros $|0\rangle^n$ to all ones $|1\rangle^n$. We then apply the angle-dependent operators $U(B, \beta_i)U(C, \gamma_i)$, one after the other, for each angle pair $\gamma_i, \beta_i$, to $s$. This is the state $|\vec{\gamma}, \vec{\beta}\rangle$. When measured, this state collapses to a bitstring $z_p$. $z_p$ is our approximate solution. The average $z_p$ produced by QAOA has an objective value close to $F_p(\vec{\gamma}, \vec{\beta}) = \langle \vec{\gamma}, \vec{\beta}|C|\vec{\gamma}, \vec{\beta}\rangle$. $F_p$ is the objective

value of the expectation for the state $|\vec{\gamma}, \vec{\beta}\rangle$. If the measured state $z_p$ has a better objective value than that of the initial solution $z$, we store $z = z_p$ and it's corresponding objective value $c_z = C(z_p)$.

We denote the best objective value attainable for a particular selection of angles $\vec{\gamma}, \vec{\beta}$ as $M_p = \max_{\vec{\gamma},\vec{\beta}} F_p(\vec{\gamma}, \vec{\beta})$. To ensure the algorithm can produce a globally optimal solution, we must prove that $\lim_{p\to\infty} M_p$ is a global maximum objective value for $C$. In the next subsection, I will prove this statement on the backs of others.

**Theory**

**Lemma 1.** The objective function $C : H \to Z$, where $H$ is a $2^n$ dimensional Hilbert space, is a linear operator.

**Proof.** $T$ is a linear operator from $V$ to $W$ if for all $x, y \in V$ and $\alpha, \beta \in \mathbb{C}$,

$$T(\alpha x + \beta y) = \alpha Tx + \beta Ty$$

Let $z_1, z_2 \in H$ and $A, B \in \mathbb{C}$. Therefore,

$$C(Az_1 + Bz_2) = \sum_{\alpha=1}^{m} C_\alpha(Az_1 + Bz_2)$$

If all $C_\alpha$ are linear,

$$\begin{aligned} C(Az_1 + Bz_2) &= \sum_{\alpha=1}^{m} AC_\alpha(z_1) + BC_\alpha(z_2) \\ &= \sum_{\alpha=1}^{m} AC_\alpha(z_1) + \sum_{\alpha=1}^{m} BC_\alpha(z_2) \\ &= A\sum_{\alpha=1}^{m} C_\alpha(z_1) + B\sum_{\alpha=1}^{m} C_\alpha(z_2) \\ &= AC(z_1) + BC(z_2) \end{aligned}$$

Therefore, $C$ is linear if all $C_\alpha$ are linear. However, if one $C_j$ is non-linear,

$$\begin{aligned}
&\therefore C_j(Az_1 + Bz_2) \neq AC_j(z_1) + BC_j(z_2) \\
&\therefore C(Az_1 + Bz_2) = \sum_{\alpha=1,\alpha\neq j}^{m} AC_\alpha(z_1) + BC_\alpha(z_2) \\
&\qquad\qquad + C_j(Az_1 + Bz_2) \\
&\qquad\qquad = A\sum_{\alpha=1,\alpha\neq j}^{m} C_\alpha(z_1) + B\sum_{\alpha=1,\alpha\neq j}^{m} C_\alpha(z_2) + C_j(Az_1 + Bz_2)
\end{aligned}$$

Assume $C$ is linear.

$$\begin{aligned}
&\therefore C(Az_1 + Bz_2) = \sum_{\alpha=1}^{m} AC_\alpha(z_1) + BC_\alpha(z_2) \\
\implies A\sum_{\alpha=1}^{m} C_\alpha(z_1) + B\sum_{\alpha=1}^{m} C_\alpha(z_2) &= A\sum_{\alpha=1,\alpha\neq j}^{m} C_\alpha(z_1) + B\sum_{\alpha=1,\alpha\neq j}^{m} C_\alpha(z_2) + C_j(Az_1 + Bz_2) \\
&= A\big(\sum_{\alpha=1}^{m} C_\alpha(z_1)\big) - AC_j(z_1) \\
&\quad + B\big(\sum_{\alpha=1}^{m} C_\alpha(z_2)\big) - BC_j(z_2) \\
&\quad + C_j(Az_1 + Bz_2) \\
&\therefore 0 = -AC_j(z_1) - BC_j(z_2) + C_j(Az_1 + Bz_2) \\
\implies C_j(Az_1 + Bz_2) &= AC_j(z_1) + BC_j(z_2)
\end{aligned}$$

However, $C_j$ is non-linear. Since this is a contradiction, $C$ must be non-linear. Without loss of generality, this argument can be extended to all clauses $C_\alpha$. Therefore, we can conclude that $C$ is a linear operator if all $C_\alpha$ are linear. Otherwise, $C$ is non-linear.

**Lemma 2.** All terms in $U(C, \gamma_i)$ commute.

**Proof.** We seek a Hamiltonian operator $H_\alpha$ which represents an arbitrary clause, $C_\alpha$. For every arbitrary clause, every bitstring $v \in \{0, 1\}^n$ is a possible solution with an objective function value $\lambda_z = C_\alpha(z)$. Since a Hamiltonian operator can only evolve to an eigenvector, and we must

be able to evolve to every bitstring to sample all solutions for a given problem, every given bitstring $v$ must be a given eigenvector of $H_\alpha$. This eigenvector will have eigenvalue $\lambda_v$. By Singular Value Decomposition, we know the eigenvectors and eigenvalues uniquely determine the operator, subject to a permutation. When $V$ is the matrix composed of eigenvectors (column vectors) horizontally stacked against oneanother, and $\Lambda$ is the diagonal matrix containing the corresponding eigenvalues (ordered same as eigenvectors),

$$H_\alpha = V\Lambda V^{-1}$$

Since all bitstrings are eigenvectors, $V$ must be some permutation of a matrix containing all bitstrings (as column vectors) horizontally stacked against one another. For convenience, let the order in which eigenvectors are placed (left to right) correspond to the integer value of the bitstring. Therefore,

$$\begin{aligned} V &= \begin{bmatrix} |0\rangle & |1\rangle & \dots & |n-1\rangle \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} = I \\ \therefore V^{-1} &= I^{-1} = I \end{aligned}$$

Hence, the required Hamiltonian operator for an arbitrary clause $C_\alpha$ is

$$\begin{aligned} H_\alpha &= V\Lambda V^{-1} \\ &= I\Lambda I \\ &= \Lambda \end{aligned}$$

The corresponding eigenvalue matrix will consist of the eigenvalues, or objective function values, for respective bitstrings in the same column. This is written as:

$$\begin{aligned}\Lambda &= \text{diag}(\lambda_{|0\rangle}, \lambda_{|1\rangle}, \ldots, \lambda_{|n-1\rangle}) \\ &= \begin{bmatrix} \lambda_{|0\rangle} & 0 & \ldots & 0 \\ 0 & \lambda_{|1\rangle} & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & \lambda_{|n-1\rangle} \end{bmatrix}\end{aligned}$$

Since $H_\alpha = \Lambda$, we know that $H_\alpha$ is a diagonal matrix. Consequently, the Hamiltonian operator representing the overall MAXSAT instance is a sum of individual, clause-specific Hamiltonian operators i.e. $H = \sum_{\alpha=1}^{m} H_\alpha$. Henceforth, $H_\alpha$ will be analogous to $C_\alpha$, as is, $H$ to $C$. When applying this to our representation of $U(C, \gamma_i)$,

$$\begin{aligned}U(C, \gamma_i) &= e^{-i\gamma_i C} \\ &= e^{-i\gamma \sum_{\alpha=1}^{m} C_\alpha} = e^{\sum_{\alpha=1}^{m} -i\gamma_i C_\alpha} \\ &= \prod_{\alpha=1}^{m} e^{-i\gamma_i C_\alpha}\end{aligned}$$

The matrix exponential of a diagonal matrix is also a diagonal matrix. Additionally, diagonal matrices commute when multiplied with one another. Therefore, all terms in $U(C, \gamma_i)$, being diagonal, commute with one another.

**Lemma 3.** Since $C$ has integer eigenvalues, $\gamma_i$ can be restricted to $[0, 2\pi]$.

**Proof.** Since $C$ is a diagonal operator, $\gamma_i C$ is diagonal. Consequently, $\cos \gamma_i C$ and $\sin \gamma_i C$ are diagonal. From [9], we can generalize Euler's identity to matrices such that,

$$\begin{aligned} e^{iC} &= \cos C + i \sin C \\ \implies e^{-i\gamma_i C} &= \cos \gamma_i C - i \sin \gamma_i C \end{aligned}$$

Using the notation $[A]_{ij}$ is the element at the $i$-th row and $j$-th column of $A$,

$$[e^{-i\gamma_i C}]_{kk} = [\cos \gamma_i C]_{kk} - i[\sin \gamma_i C]_{kk}$$

The exponential, sine and cosine of a diagonal matrix are diagonal and constructed by taking the exponential (, cosine, or sine, respectively) of every diagonal entry in $C$. Therefore,

$$[e^{-i\gamma_i C}]_{kk} = \cos \gamma_i [C]_{kk} - i \sin \gamma_i [C]_{kk}$$

To prove $\gamma_i$ can be restricted to $[0, 2\pi]$,

$$\begin{aligned}[e^{-i(\gamma_i+2\pi)C}]_{kk} &= \cos{(\gamma_i + 2\pi)}[C]_{kk} - i \sin{(\gamma_i + 2\pi)}[C]_{kk} \\ &= \cos(\gamma_i [C]_{kk} + 2\pi[C]_{kk}) - i \sin(\gamma_i [C]_{kk} + 2\pi[C]_{kk})\end{aligned}$$

Since the eigenvalues of $C$ are integers, and $C = H = \sum_{\alpha=1}^{m} \Lambda_\alpha$ is a diagonal matrix, $[C]_{kk}$ is an integer. Cosine and sine both have period $2\pi$. Since the input angle is the sum of $\gamma_i [C]_{kk}$ with an integer multiple of $2\pi$,

$$\begin{aligned}[e^{-i(\gamma_i+2\pi)C}]_{kk} &= \cos \gamma_i [C]_{kk} - i \sin \gamma_i [C]_{kk} \\ &= e^{-i\gamma_i C} \\ \implies e^{-i(\gamma_i+2\pi)C} &= e^{-i\gamma_i C}\end{aligned}$$

Hence, $e^{-i\gamma_i C}$ is a periodic function with period $2\pi$. Therefore, $\gamma_i$ can be restricted to $[0, 2\pi]$.

**Lemma 4.** All terms in $U(B, \beta_i)$ commute.

**Proof.** $B$ is already the sum of any commuting, single bit, operators, $\sigma_j^x$ for the $j$-th bit. For our purposes, let $\sigma^x$ be the Pauli X operator. Where $\otimes$ is the Kronecker product, these commuting

operators are written as:

$$I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\sigma^x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$\sigma_j^x = I_2 \otimes \ldots j - 2 \text{ times } \cdots \otimes I_2 \otimes \sigma^x \otimes I_2 \otimes \ldots n - j - 2 \text{ times } \cdots \otimes I_2$$

Using the representation for $U(B, \beta_i)$,

$$\begin{aligned} U(B, \beta_i) &= e^{-i\beta_i B} \\ &= e^{-i\beta_i \sum_{j=1}^{n} \sigma_j^x} = e^{\sum_{j=1}^{n} -i\beta_i \sigma_j^x} \\ &= \prod_{j=1}^{n} e^{-i\beta_i \sigma_j^x} \end{aligned}$$

When two operators $A, B$ commute, their matrix exponentials commute as well i.e.

$$AB = BA \rightarrow e^A e^B = e^B e^A$$

Hence, all terms in $U(B, \beta_i)$ commute.

**Lemma 5.** $\beta_i$ can be restricted to $[0, 2\pi]$.

**Proof.** We know that

$$U(B, \beta_i) = \prod_{j=1}^{n} e^{-i\beta_i \sigma_j^x}$$

Using the formula derived in [10],

$$e^{-i\beta_i \sigma_j^x} = I_2 \otimes \cdots \otimes I_2 \otimes e^{-i\beta_i \sigma^x} \otimes I_2 \otimes \cdots \otimes I_2$$

From the result in [11], the exponential of the $\beta_i$-dependent Pauli X operator can be written

as

$$e^{-i\beta_i\sigma^x} = I\cos\beta_i + i\sigma^x\sin\beta_i$$
$$\implies e^{-i\beta_i\sigma_j^x} = I_2 \otimes \cdots \otimes I_2 \otimes (I\cos\beta_i + i\sigma^x\sin\beta_i) \otimes I_2 \otimes \cdots \otimes I_2$$

Since cosine and sine have period $2\pi$, we can see that $\beta_i$ can be restricted to $[0, 2\pi]$.

**Lemma 6.** $|\vec{\gamma}, \vec{\beta}\rangle$ can be produced by a quantum circuit of depth at most $mp + p$.

**Proof.** Every $U(C, \gamma_i)$ counts as $m$ operators since they are the product of $m$ clause-specific Hamiltonian operators. Hence, a circuit of depth $m$ is required to produce one $U(C, \gamma_i)$. Since $U(B, \beta_i)$ is the product of $n$ exponentiated, commuting, single bit operators whose overall form can be evaluated within constant depth, a circuit with constant depth, 1, is required to produce one $U(B, \beta_i)$.

Within one step of QAOA, $U(C, \gamma_i)$ and $U(B, \beta_i)$ are applied one after another. This requires a circuit depth of $m + 1$, neglecting some constant factor for $U(B, \beta_i)$. Since $p$ applications of $U(C, \gamma_i)U(B, \beta_i)$ are performed, the depth of the circuit which produces $|\vec{\gamma}, \vec{\beta}\rangle$ is at most $p(m + 1) = mp + p$.

**Lemma 7.** The maximization of $F_p(\vec{\gamma}, \vec{\beta})$ over $\vec{\gamma}, \vec{\beta}$ can be viewed as a constrained maximization at $p$ so

$$M_p \geq M_{p-1}$$

**Proof.** When $p$ is increased by 1, one is able to attain the same expected objective function value by setting the new $\gamma_p, \beta_p$ to zero. Therefore, the maximization of the expectation value for a smaller number of angles can be viewed as a constrained version of the same problem for a larger number of angles case, such that all extra angles are set to zero. Hence, a lower bound on the maximized expected objective function value over all permutations of angles $M_{p+1}$ is at least $M_p$. Using a shift in index, $M_p \geq M_{p-1}$.

**Statement 1.** Quality of Approximation improves with $p$.

Using Lemma 7, we know that if our angles are selected wisely, we are guaranteed to

keep increasing the maximized expectation of the objective function value for a sampled $|\vec{\gamma}, \vec{\beta}\rangle$. Intuitively, we should be able to say something about the expectation value over all choices of angles for different values of $p$. In other words, it should be rational to predict that $F_p$ will increase with $p$. However, it is difficult to prove this statement over all choices of $\vec{\gamma}$ and $\vec{\beta}$ without more knowledge of the problem being solved.

**Statement 2.** $\lim_{p\to\infty} M_p = \max_{z\in\{0,1\}^n} C(z)$

Informally, this can be proven using the relationship between QAOA and Quantum Annealing. If we increase each subsequent $\gamma_i$ in $\vec{\gamma}$ by an infinitesimally small amount from 0 to $2\pi$, while decreasing each corresponding $\beta_i$ by an equally small amount from $2\pi$ to 0, QAOA can approximate the tuning from the simple $\beta$-dependent Hamiltonian to the complex, $\gamma$-dependent, problem Hamiltonian. Therefore, as $p$ grows larger, we are able to better approximate this infinitely slow cooling. Hence, as $p$ goes to infinity, the maximized expectation value $M_p$ for the sampled state will approach the global maxima, just like Quantum Annealing.

**Statement 3.** Depth of circuit grows linearly with $p$ times (at worst) the number of constraints.

Since the most time-consuming portion of QAOA comes from generating $|\vec{\gamma}, \vec{\beta}\rangle$, using Lemma 6, the worst case circuit depth of QAOA will be $O(mp + p) = O(mp)$. Therefore, the depth of the QAOA circuit increases linearly with $p$ times the number of constraints $m$.

**Criticism**

The most vague procedure is picking $\vec{\gamma}, \vec{\beta}$ such that $F_p(\vec{\gamma}, \vec{\beta})$ is close to $M_p$. However, there is no "one-size-fits-all" approach. One can perform a classical search over $\vec{\gamma}, \vec{\beta}$ using some gradient based solver (since the partial derivatives of $F_p$ are quadratic at most). However, this sampling requires multiple calls to the quantum computer with multiple sets of angles until the best is found. The second procedure makes use of structure in $C$. If $C$ is an instance of MAXSAT, an NP-Complete problem, we are able to reduce $F_p$ to a sum over sub-problems (or sub-graphs) in $C$. However, computing the terms in this sum requires doubly exponential time when the number of sub-problems grows too large. For large $n$, this second approach would be at least as slow as classical solvers.

One practical consideration can be found within the coherence times of Noisy Intermediate-Scale Quantum Computers (NISQ): if your computation runs too long, susceptibility to environmental noise increases and errors build up. This can cause serious experimental difficulties. Especially if $p$ increases, more operators are required to form the state $|\vec{\gamma}, \vec{\beta}\rangle$, increasing running time. Given small coherence times, large $p$ may provide junk data due to noise. Large $p$ are necessary to find global maxima of $C$.

# CHAPTER III

# CONCLUSION

QAOA is a tunable optimization algorithm for $NP$-complete problems. The quantum circuit which implements QAOA does so with a depth which does not exceed (at worst) $p$ times the number of constraints $m$. The mean objective function value of the sampled solutions, maximized over selection of angles, $M_p$, increases with the tuning parameter for the quality of the approximation, $p$. Furthermore, as the number of angles goes to infinity, QAOA is guaranteed to provide a globally optimal solution.

Early results indicate QAOA is able to provide optimal solutions rivalling classical optimizers. However, whether the speedup is worth the effort remains to be seen. [12] presents a whole class of classical, global optimization algorithms who are "at least as promising as QAOA for approximate optimization".

However, the selection of angles $\vec{\gamma}, \vec{\beta}$ is paramount and requires careful attention. Either automatic, gradient-based optimizers, or, manual selection from a course grid of possible pairs of angles is needed to produce $2p$ angles for which $F_p(\vec{\gamma}, \vec{\beta})$ is maximized. This presents a great avenue for future research.

# REFERENCES


[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd ed., 2009.

[2] K. H. Rosen, *Discrete Mathematics and Its Applications*. McGraw-Hill Higher Education, 5th ed., 2002.

[3] R. Feynman, "There's plenty of room at the bottom. talk given at the annual meeting of the american physical society at caltech," 1959.

[4] N. D. Mermin, *Quantum Computer Science: An Introduction*. USA: Cambridge University Press, 2007.

[5] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Comput.*, vol. 26, p. 1484–1509, Oct. 1997.

[6] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, (New York, NY, USA), p. 212–219, Association for Computing Machinery, 1996.

[7] S. Geman and D. Geman, "Stochastic relaxation, gibbs distributions, and the bayesian restoration of images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no. 6, pp. 721–741, 1984.

[8] E. Farhi, J. Goldstone, and S. Gutmann, "A Quantum Approximate Optimization Algorithm," *arXiv e-prints*, p. arXiv:1411.4028, Nov 2014.

[9] G. Argentini, "A matrix generalization of euler identity $e^{ix} = cosx + isinx$," 2007.

[10] Y. Takeuchi, "Calculus of exponential matrix with kronecker product," *IEEJ Transactions on Fundamentals and Materials*, vol. 111, no. 7, pp. 680–681, 1991.

[11] F. D. Zela, "Closed-form expressions for the matrix exponential," *Symmetry*, vol. 6, no. 2, p. 329–344, 2014.

[12] M. B. Hastings, "Classical and quantum bounded depth approximation algorithms," 2019.